AUTOMATIC GENERATION OF SIMULATION MODELS FROM NEUTRAL LIBRARIES: AN EXAMPLE

Young Jun Son

Industrial and Manufacturing Engineering
The Pennsylvania State University
University Park, PA 16802, U.S.A.

Albert T. Jones

Manufacturing Systems Integration Division
National Institute of Standards & Technology
Gaithersburg, MD 20899, U.S.A.

Richard A. Wysk

Industrial and Manufacturing Engineering
The Pennsylvania State University
University Park, PA 16802, U.S.A.

ABSTRACT

Researchers at the National Institute of Standards and Technology have proposed the development of neutral libraries of simulation components. The availability of such libraries would simplify the generation of simulation models, enable component-based modeling, and speed Internet-based simulation services. The result would be a reduction in the complexity of simulation modeling and analysis. In this paper, we consider a discrete-event simulation of the flow of jobs through a job shop. We describe the information requirements for the components in that simulation and provide formal models based on those requirements. We then derive a database structure from these formal models and discuss the population of that database with the data entries for a sample job shop. Finally, we examine the translators we developed to go from the neutral representation of the simulation components to the representation required by a commercial simulation package.

1 INTRODUCTION

Simulation has been a useful design and analyses tool used to model manufacturing systems for decades. A number of commercial products, with a range of capabilities and price tags, are on the market. Each of these packages has its own user interface for building models, animation capabilities for viewing the evolution of models over time, and tools for analyzing the output from those models. The degree of difficulty in building models, the fidelity of the visualization, and the sophistication of the analysis tools vary dramatically. Consequently, building, running, and analyzing a simulation model can be a very time-consuming and error-prone process.

To address the model building issue, researchers at the National Institute of Standards and Technology (NIST) have proposed the development of neutral libraries of simulation components and model templates. The former would contain detailed, formal, information models of all commonly used simulation components - queues, machines, transporters, and so forth. Each of these component models would have views tailored to specific modeling scenarios. These scenarios would be defined by different modeling templates - such as an equipment simulation, a material flow simulation, a supply chain simulation, and so forth. The availability of such libraries, together with the requisite translators, would simplify the model-building process. It would also enable component-based modeling, model reuse, and Internet-based services, all of which could reduce the complexity and effort of simulation in manufacturing (see Figure 1).

After the library of simulation objects is constructed, each component in the library becomes a basic building block (module) to model systems of interest. Then, a translator, which we call a model builder (see Figure 1), will generate a simulation model for a specific commercial package from the neutral descriptions of the components. In this research, model builders for Arena and ProModel have been designed and implemented[1]. Each model builder generates a model for the specified simulation language.

[1] Certain commercial software products are identified in this paper. These products were used only for demonstrations purposes. This use does not imply approval or endorsement by NIST, nor does it imply that these products are necessarily the best available for the purpose.

Consequently, there are differences between the two we have built.

In this paper, we discuss the model builder for ProModel - the model builder for Arena and a comparison between the two will be presented in another paper. In Section 2, we describe the simple job shop that serves as our example manufacturing system. The simulation will model the flow of jobs through that job shop. In Section 3, we provide the information requirements for the simulation components. In Section 4, we include a formal, information model for some of those requirements, show the resulting database structure, and discuss the population of that database with the data entries for our sample job shop. Finally, we examine the model builder, translator, we developed to go that database to representation in ProModel.

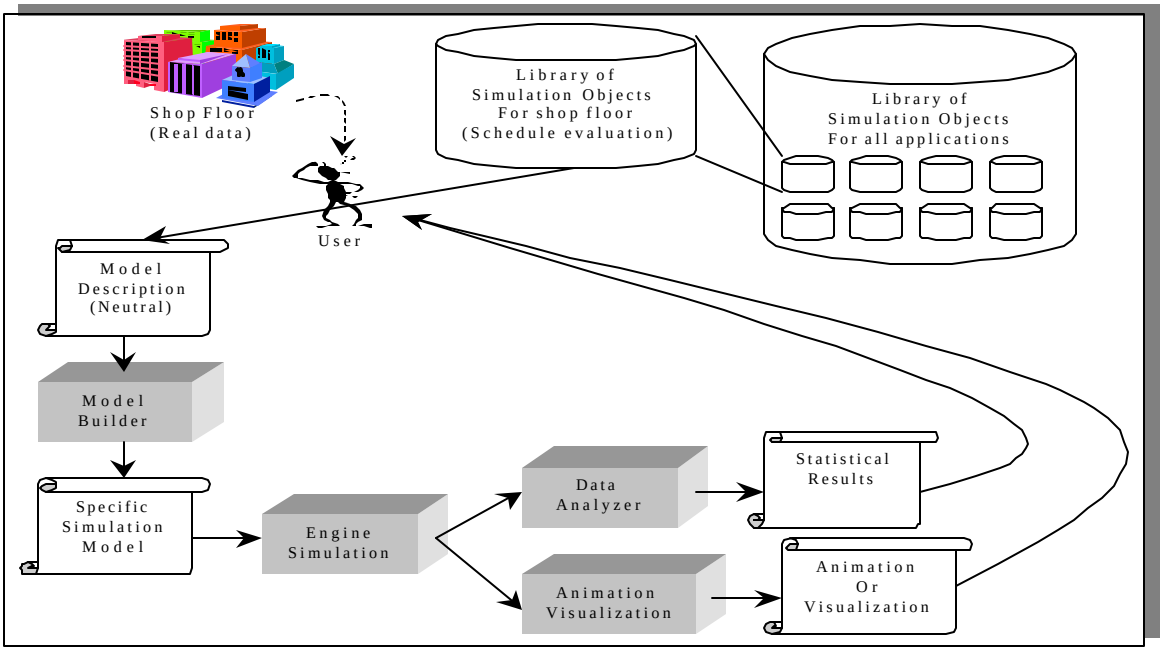


Figure 1: New concept using library components

2 EXAMPLE MANUFACTURING SYSTEM

An example shop floor is shown in Figure 2. It contains a system input buffer, a system output buffer, and three processing stations - penn1, penn2, and penn3. Station penn1 has a dedicated input buffer, penn1_in, and a dedicated output buffer, penn1_out. Station penn2 has a dedicated input buffer only, penn2_in, and penn3 has neither input nor output buffers. The capacity of system input and output buffers is 100, while the capacity of each processing station is 1. This shop can make three different products: a mouse, a notebook, and a pen. The mouse requires processing at penn1 and penn2. The notebook requires processing at penn2 only, and the pen requires processing at penn3 only. Production of one item of each product will be demonstrated in this paper.

3 INFORMATION REQUIREMENTS

For the flow simulation used in this project, there are six classes of objects: header information, experiment information, shop floor information, product-process information, production information, and output information.

3.1 Header Information

The header information object provides the introductory information about the simulation file. Each simulation file has exactly one header information object. The header information object is composed of a simulation file name, an analyst name, a layout file name, a save date, a save time, and a description.

- Simulation file name - The unique name for the simulation file.
- Analyst name - The name of person that created the simulation file.

- Saved date - The date of the creation of the simulation file.
- Saved time - The time of the creation of the simulation file.
- Description - A word or group of words that describe the basic information of the simulation file.
- Layout file name - The name for the layout file that is used for simulation background.

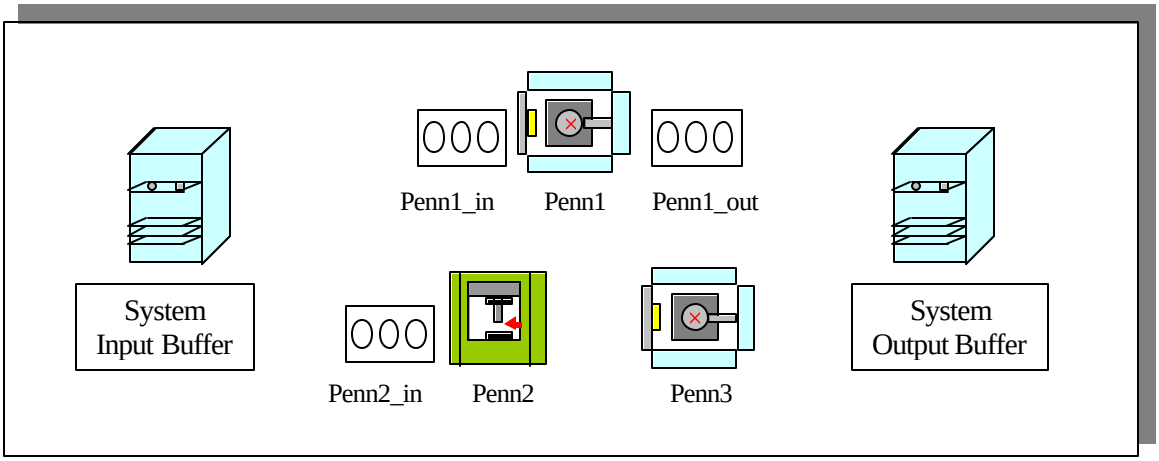


Figure 2: Example job shop system

3.2 Experimental Information

Each simulation file is associated with one experimental information object. The experimental information object describes the environmental setting to run the simulation file and obtain the requested performance measures. That environment includes a time unit, a distance unit, a beginning time, a replication time, the number of replications, a terminating condition, and a collection of output information objects. Each output object is composed of a performance measure name.

- Time unit - The unit of time in the simulation file, usually one of day, hour, minute, or second.
- Distance unit - The unit of distance in the simulation file, usually a meter or a foot.
- Beginning time - The real value defining the beginning time of the first replication.
- Replication time - The length of each replication.
- Number of replications - The integer value defining the number of replications to be executed.
- Terminating condition - An optional field to specify the terminating condition. If nothing is specified in the filed, the simulation runs until the replication time.
- Output information - An optional field to specify the performance name of interest.

3.3 Shop Floor Information

The shop floor information object describes the physical entities on the shop floor. Each simulation file is associated with one shop floor information object. The shop floor information object is composed of a set of station information objects. Each station information object is composed of a station name, a capacity, a description, and a station type information object. The station type information object is associated with either a processing station information object or a buffer station information object. The processing station information object is composed of an optional station buffer name and an optional station output buffer name. In addition, the buffer station information object is defined by a buffer type item. In the actual shop floor, there are two classes of stations: a processing station and a buffer station. If the station information object does not include the processing station information, it is interpreted as a buffer station.

- Station name - The unique name for the station.
- Capacity - The integer value defining capacity characteristics of the station.
- Station type - Unique identifier for station; e.g. processing, buffer.
- Processing station information
  - Station input buffer name - The optional string field containing station name for the dedicated input buffer in the processing station. Note the input station input buffer needs to be defined explicitly as a station information.

- Station output buffer name - The optional string field containing station name for the dedicated output buffer in the processing station. Note the output station input buffer needs to be defined explicitly as a station information.

- Buffer station information
  - Buffer type items - The string field specifying the types of buffers. It contains four values: system_input_buffer, system_output_buffer, station_input_buffer, and station_output_buffer.

3.4 Product/Process Information

The product/process information object provides the run time data for a simulation. It is composed of a product name and a process plan information object. The process plan object is composed of a process plan name and an ordered list of operation information objects. Each operation object is composed of an operation number, an operation name, a description, a station name, a processing time, a next station name, and a routing time.

- Product name - The unique name for the product.
- Process plan information
  - Process plan name - The unique name for the process plan.
  - Operation information
  - Operation number - The unique identifier for the operation.
  - Operation name - The word or a group of words defining the current operation.
  - Station name - The name of station where the operation occurs.
  - Processing time - The real value defining the duration taken for the current operation. If the station is associated with a processing station, this value is associated with actual machining time.
  - Next station name - The name of station that the job will visit next.
  - Routing time - The real value defining the duration taken moving from the current station to the next station. In a more complete modeling, an equipment will be involved such as an AGV or a conveyor. Therefore, equipment contention will be included in the model. In the current version of the document, the material transporters are not included in the model, and it is assumed that jobs can move to next station whenever there is available capacity.
  - Description - A word or a group of words that describe the operation.

3.5 Production Information

A production information object provides the data for what is produced in the simulation and associated due dates. The production information is composed of a product name and a set of job information objects. The attributes for each job information object include a job name, a quantity, an arrival time, and a due time.

- Product name - The name of product that the job is associated with. Using this filed, we can derive the associated process plan, which is provided by the product process information object in Section 2.4.
- Job information
  - Job name - The unique name of a job. A job is an atomic object associated with one product.
  - Quantity - The number of products to be produced.
  - Arrival time - The time when the current job arrives in the simulation file.
  - Due time - The time by when the current job is wanted to be finished by the customer.

3.6 Output Information

The simulation output is stored in a returned result information object. Each such object is composed of a performance measure name, and an associated graph name.

- Performance measure name - The unique name of the performance name. The contents of this filed will be specific to the commercial simulation packages unless a generic way of specifying the name is created. This filed needs to be associated with the performance measure name for the output information object.
- Graph name - The name of graph associated with a particular performance measure. This attribute is optional since not all performance measures will be represented as a graph. In general, each performance measure can have many different types of graphs. In this research, however, we only allowed one type for each performance measure.

4 INFORMATION MODELING

Based on the preceding information requirements, a complete information model has been developed in EXPRESS [2, 4]. Due to limited space, we show only part of that model.

4.1 Schema

Types, entities, and functions has been defined formally as follows:

```
SCHEMA discrete_event_simulation;
TYPE buffer_type_items = ENUMERATION OF
```

```
    (station_input_buffer,
     station_output_buffer,
     system_input_buffer,
     system_output_buffer);
END_TYPE;

TYPE name = STRING;
END_TYPE;

TYPE station_type_information = SELECT
    (processing_station_information,
     buffer_station_information);
END_TYPE;

ENTITY shop_floor_information;
    station_data : SET [0:?] OF station_
information;
END_ENTITY;

ENTITY station_information;
    station_name : name;
    capacity : INTEGER;
    station_type_data : station_type_
information;
    description : OPTIONAL text;
 UNIQUE
    UR1:    station_name;
END_ENTITY;

ENTITY buffer_station_information;
    buffer_type : buffer_type_items;
END_ENTITY;

ENTITY processing_station_information;
    station_input_buffer_name  :  OPTIONAL
name;
    station_output_buffer_name : OPTIONAL
name;
END_ENTITY;

-   - - - -

END_SCHEMA;
```

4.2 Database Instantiation

From the schema in the previous section, we generated a collection of database tables in MS Access 97 (see Figure 3). The tables in the figure belong to two classes. The first class contains a table for each entity in the EXPRESS schema. The second class contains tables that specify the relationship among the entities.

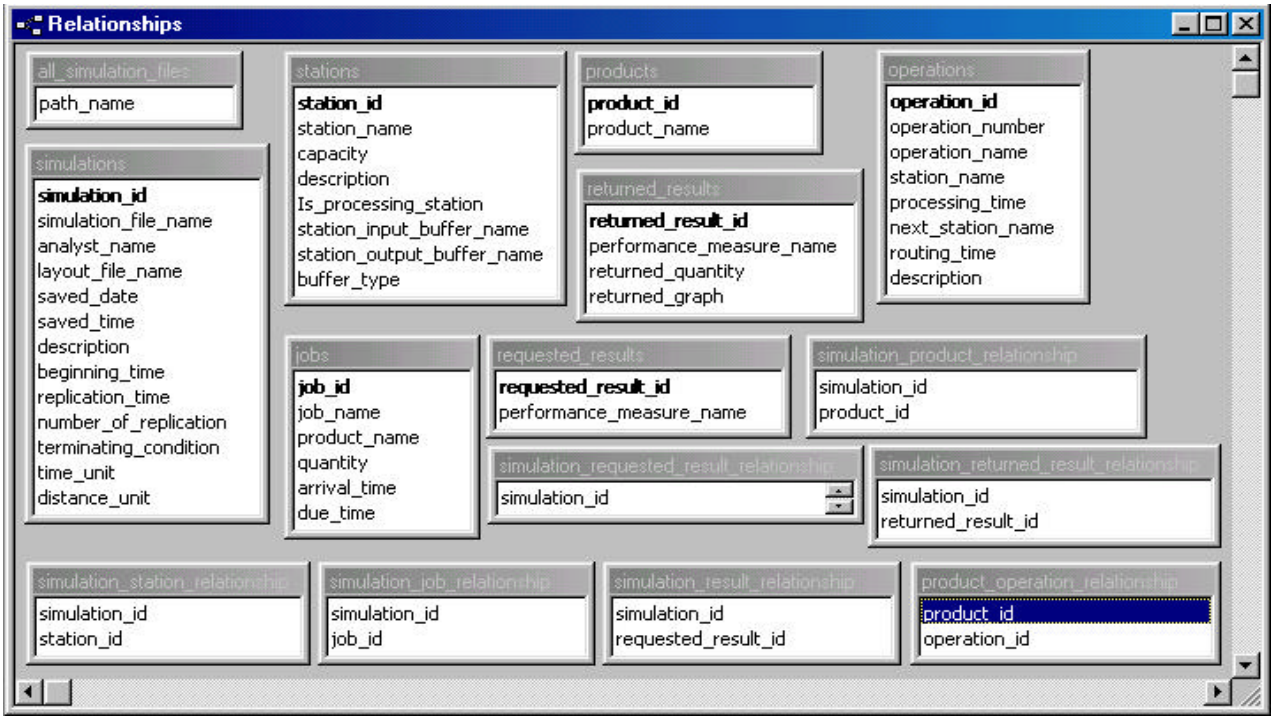


Figure 3: Database tables associated with the formal EXPRESS model

4.3 Database for Example System

Several database tables associated with the example system in Figure 2 have been populated by hand and are shown in Figure 4. In the following sections, we describe how we generated the corresponding ProModel .mod file.

simulations : Table

simu	simulation_file_name	analyst_name	begin	replication_time	number_of_replication	time_unit
1	C:\temp\myfile	Young Jun SON	0	1000	1	minute
2	C:\temp\secondfile	Eunkoo Lee	0	2500	1	second

Record: 1 of 2

stations : Table

station_id	station_name	capacity	ls_processing	station_input	station_output	buffer_type
1	penn_in_storage	100	☐			system_input_buffer
2	penn1_in	3	☐			station_input_buffer
3	penn1	1	☑	penn1_in	penn1_out	
4	penn1_out	3	☐			station_output_buffer
5	penn2_in	3	☐			station_input_buffer
6	penn2	1	☑	penn2_in		
7	penn3	1	☑			
8	penn_out_storage	100	☐			system_output_buffer

Record: 1 of 8

a: Simulation header, experiment, and station information for the example system

operations : Table

operation_id	operation_number	station_name	processing_time	next_station_name	routing_time
1	10	penn_in_storage	0	penn1	50
2	20	penn1	300	penn2	50
3	30	penn2	200	penn_out_storage	50
4	40	penn_out_storage	0		
5	10	penn_in_storage	0	penn1	50
6	20	penn1	450	penn_out_storage	50
7	30	penn_out_storage	0		
8	10	penn_in_storage	0	penn2	50
9	20	penn2	280	penn3	50
10	30	penn3	370	penn_out_storage	50
11	40	penn_out_storage	0		0

Record: 1 of 11

products : Table

product_id	product_name
1	penn_mouse
2	penn_notebook
3	penn_pen

Record: 1

product_operation_relatio...

product_id	operation_id
1	1
1	2
1	3
1	4
2	5
2	6
2	7
3	8
3	9
3	10
3	11

Record: 1

jobs : Table

job_id	job_name	product_name	quantity	arrival_time	due_time
1	job1	penn_mouse	1	30	1700
2	job2	penn_notebook	1	15	1750
3	job3	penn_pen	1	25	1800

Record: 1 of 3

b: Process plan, product, and job information for illustration

requested_results : Table

requested_result_id	performance_measure_name
1	Davg(penn1_Busy)
2	Davg(penn2_Busy)

Record: 2 of 2

simulation_requested_result_rel...

simulation_id	requested_result_id
1	1
1	2

Record: 2 of 2

c: Requested_results information for illustration

Figure 4: Database information for the example system [y1]

5 DESIGN OF MODEL BUILDER

The role of the model builder is to create a discrete-event simulation model from the neutral description of the system and the actual data in the database. In that sense, the model builder serves as a translator. The following sections describe how that translator works.

5.1 Shop Floor

The first step in creating the simulation model is the construction of the shop floor. The model builder creates this shop floor from the "stations" table in the database (see Figure 4-a). Each station in that table is associated with a "location" template in Promodel (see Figure 5-a). The data for first two columns in this template come directly from the stations table. The remaining columns in the template are defaults. The use of the remaining data in the locations table is described in Section 5.2.

In general, job shops operate in one of two modes: push and pull. Push modes implies there is a predetermined schedule that jobs will follow through the shop. Pull mode implies that jobs go through without such a schedule. In our example, there is no schedule. Therefore, we need to implement a pull mode. Promodel provides the pull capability by default, so no pull logic is defined.

5.2 Job Flow Through the Shop

The shop floor was constructed so that any possible routings and processing times can be implemented. To control the flow of jobs through the shop during a particular run, Promodel requires explicit values for the routings and processing times. Variables for these values are contained in the process template and the routing template; exact values are contained in the initialization file, (see Figures 5-a, 5-b).

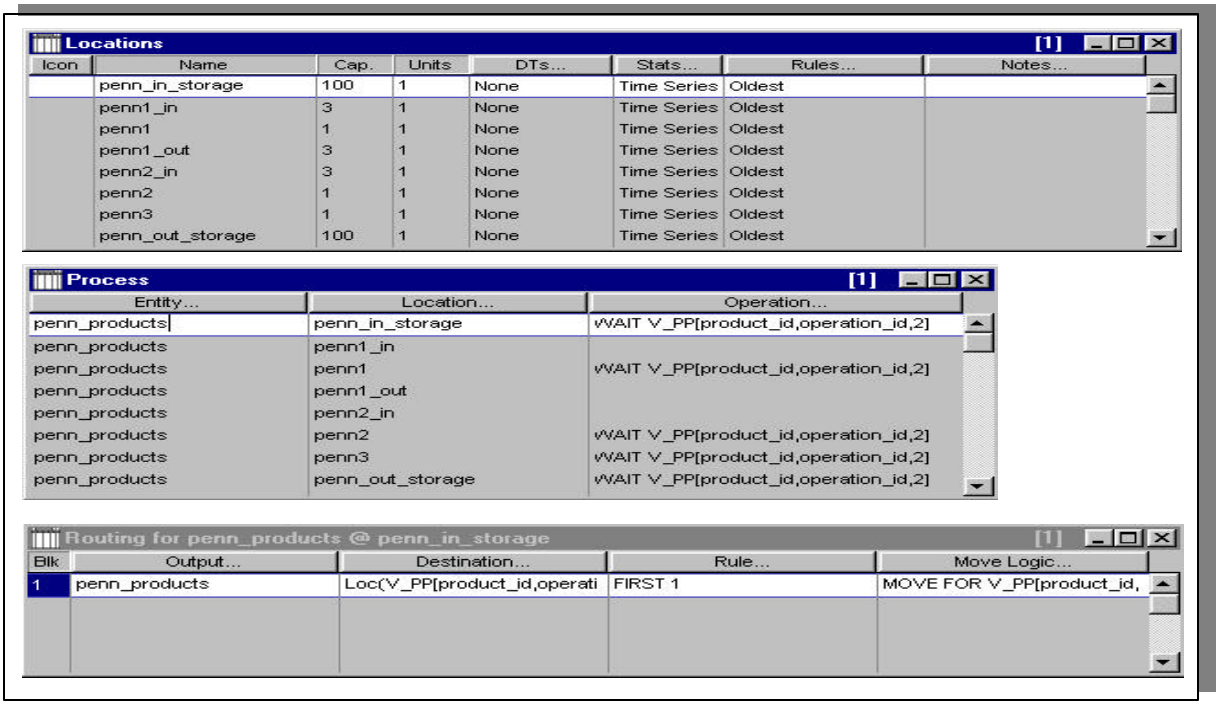
The initialization file contains the data for each specific run of the simulation. The model builder is designed so that the system is data-driven, the same model can be run many times by simply changing this file. The initialization file contains process plan data, which is collection of 3-dimensional arrays (product_id, operation_id, n) where n =1,…,5. The meanings of the 5 values of n are: current location, processing time at this location, next processing station, travel time, next physical location. The actual data values are derived from the stations table and the operations table in the database.

For each Entity-Location pair, the process template specifies a 3-dimensional variable called Operation. The first entry is the product_id, the second is the operation_id, and the third is the processing time. These entries are read in from the process plan part of the initialization file, Figure 5-b. For example, the processing time for the product-id=1, the mouse at penn1, which performs operation-id=2, is 300; the processing time at penn2, which performs operation-id=3, is 200. Note that whenever the Location is a buffer, the processing time is 0.

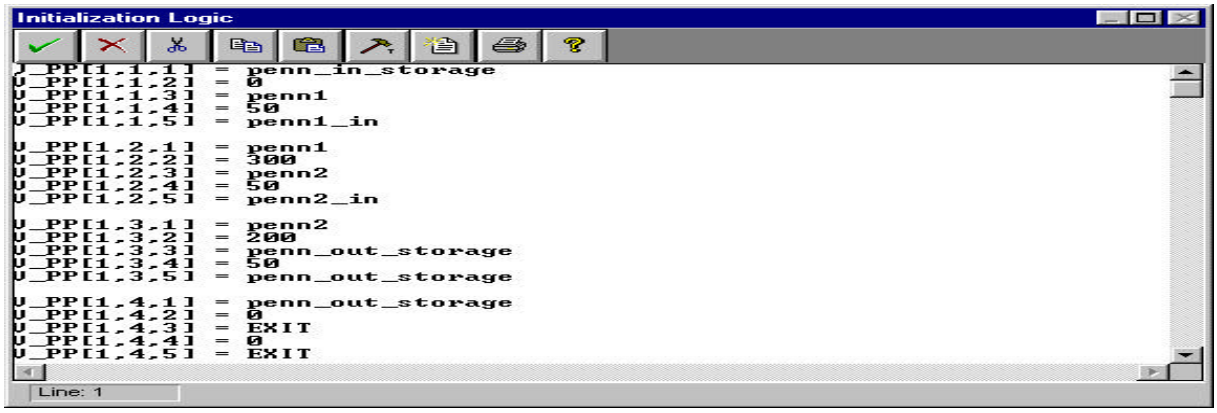
The routing template contains variables called Output, Destination, Rule, and MoveLogic. Output has the same value as Entity, unless there is an assembly operation. Rule specifies the order in which jobs are removed form each queue; in this example it is defaulted to first-in-first-out. For each location in the process template, Destination specifies the next physical location that the Entity will visit on its path through the shop and MoveLogic specifies the travel time to that location. These values are read in from the Initialization file as attributes 5 and 4, respectively. From Figure 4-b, we see that the mouse's route is input station, penn1, penn2, and output station. However, from the stations table, we see that the actual physical path is input station, penn1_in, penn1, penn1_out, penn2_in, penn2, and output station. As noted above, this physical path, including all buffers, must be represented in ProModel. This is accomplished using the routing template.

5.3 Job Arrival Information

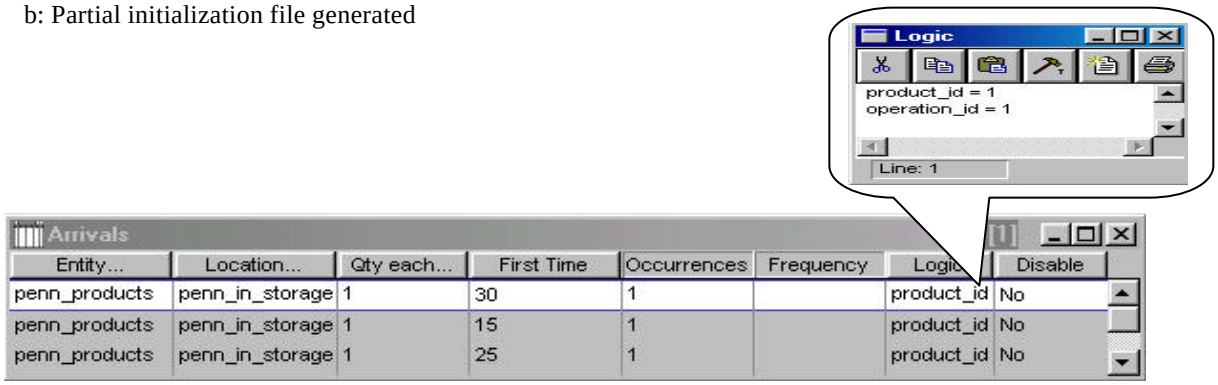
Job arrival data is contained in the arrivals template, which contains variables called Entity, Location, Qty each, First Time, Occurrences, Frequency, Logic, and Disable. An Entity arrives at Location. Qty each is the quantity of entities that arrive at each arrival time. First Time is the time of the first arrival, Occurrences is the number of occurrences for every simulation run, and Frequency is the time between arrivals. Logic is used to define any arrival logic to be executed by each entity when it is created. Disable is used to specify whether we want to temporarily disable this arrival without deleting it. Each row in the arrivals template (see Figure 5-c) is associated with one job in the "jobs" table in the database (see Figure 4-b). Entity, First Time, and Qty each are read from the "jobs" table. Location is set to penn_in_storage since all the jobs are assumed to arrive at the system input storage. Occurrences is also set to 1. The model builder is designed so that Logic assigns values read from the "jobs" table in the database (see Figure 4-b) for product_id, operation_id, and due_time. Frequency is left blank since each entity in a row is created only once for each simulation run. Frequency logic can be implemented using First time and Occurrences. For example, a job, whose First time is zero, Occurrences is 2, and Frequency is 10, is identical with two jobs, where Occurrences of each job is 1 and First time of each job is zero and 10 respectively. In this paper, the latter logic is used. Finally, Disable is set to No by default.



a: Locations, processes, and routing templates generated



b: Partial initialization file generated



c: Arrivals template generated

Figure 5: ProModel templates model for the example system[y2]

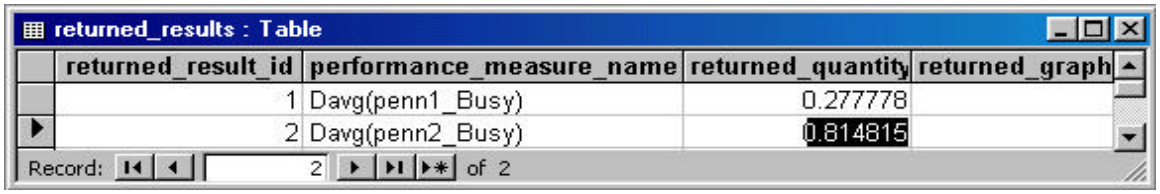


Figure 6: Results after simulation run

5.4 Simulation Result Information

Performance names of interest have been provided in Figure 4-c. Davg(location_Busy) represents the utilization of resource. The model builder understands this predefined name for performance measures. After simulation model has been run, the results have been stored in returned_results table (see Figure 6).

5.5 Implementation

The model builder has been implemented in Visual Basic 5.0. The model builder interacts with MS Access databases though the Microsoft Access 8.0 Object library and the DAO 3.5 (Data Access Objects) Object library. DAO is an application program interface (API) available with Microsoft's Visual Basic that lets a programmer request access to a Microsoft Access database. The model builder can recognize templates and objects in ProModel through the Promodel 1.0 Type library. Visual Basic 5.0 provides an environment in which we can link necessary external libraries, Microsoft Access 8.0 Object library, DAO 3.5 (Data Access Objects) Object library, and Promodel 1.0 Type library.

6 CONCLUSION

In this paper, we used a simple manufacturing example to demonstrate the use of neutral component libraries to generate simulation models in specific simulation languages. We included the information requirements for these components, as well as a partial, EXPRESS information model. An MS Access database based on the EXPRESS model and sample data have been instantiated. A model builder has been designed to generate Promodel models from such a database. Given the example manufacturing system along with artificial product, process, and order data, a complete Promodel model was generated, run, and results reported. Future research will first consider time distributions to add stochastic behaviors to the system. In addition, material-handling equipment will be included in the future research to make simulations more realistic.

ACKNOWLEDGEMENTS

This work was done as part of the intelligent manufacturing systems (IMS) MISSION project (www.ims.org), which is building an integrated modeling and simulation platform for extended enterprises and virtual enterprise networks.

REFERENCES AND BIBLIOGRAPHIES

[1] http://www.promodel.com.

[2] ISO 10303-11:1994(E), Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 11: The EXPRESS Language Reference Manual.

[3] Promodel Corporation, Promodel Version 4.1 User's Guide: Manufacturing Simulation Software, Orem, Utah, 1998

[4] Schenck, D., and Wilson, P. "Information Modeling the EXPRESS Way," Oxford University Press, New York, NY, 1994.

AUTHOR BIOGRAPHIES

YOUNG JUN SON is a graduate student and research assistant in the Department of Industrial and Manufacturing Engineering at Penn State University. He received his B.S.I.E with honors from POSTECH in 1996 and M.S.I.E from Penn State in 1998. His interests include simulation based shop floor control and automatic models generation. He was the Rotary International Multi-Year Ambassadorial Scholar in 1996 and the Council of Logistics Management Scholar in 1997. He was the representative of the Department of I&ME for the Engineering Graduate Student Council at Penn State in 1997. He is a member of IIE and SME. His email and web addresses are <yxs5@psu.edu> and <www.personal.psu.edu/yxs5>.

DR. ALBERT T. JONES is currently heading up projects at the National Institute of Standards and Technology (NIST) to investigate the functional and integration requirements for the next generation simulation tools. Prior

to this assignment, Dr. Jones spent several years as Deputy Director of the Automated Manufacturing Research Facility at NIST. During that time, Dr. Jones worked with various NIST and academic researchers on system architectures for shop floor control, cell control, and distributed scheduling. He received his MS in Mathematics and Ph.D. in Industrial Engineering from Purdue University. Dr. Jones is currently on the Executive Boards for the Winter Simulation Conference and the Engineering School at Loyola of Baltimore. He is Manufacturing Editor for several leading journals. He has Chaired or Co-chaired several international conferences, and has served on several proposal evaluation panels for NSF, NIST, and ARPA. His email and web addresses are <albert.jones@nist.gov> and <www.mel.nist.gov/msidstaff/jones.albert.htm>.

DR. RICHARD A. WYSK is well-known for his work in computer integrated manufacturing, computer automated manufacturing, computer aided process planning and concurrent engineering. He holds the Leonhard Chair in Engineering at Penn State University. Prior to his current position, he was director of the Institute for Manufacturing Systems and holder of the Royce Wisenbaker Chair in Innovation at Texas A&M. Dr. Wysk also served on the faculty of Virginia Tech and worked in industry as a research analyst for the Caterpillar Tractor Company and as production control manager for General Electric. He is a decorated Vietnam veteran. Dr. Wysk is the author of several textbooks. Honors recognizing his research include the Institute of Industrial Engineers, David F. Baker Distinguished Research Award, and the Society of Manufacturing Engineers Outstanding Young Manufacturing Engineer Award. Dr. Wysk holds Bachelor's and Master's degrees in Industrial Engineering and Operations Research from the University of Massachusetts and a Ph.D. in Industrial Engineering from Purdue University. His email and web addresses are <rwysk@psu.edu> and <www.engr.psu.edu/cim/wysk.htm>.

Page: 6

[y1]Add requested results table in the figure; for example, time in the system or time in the queue.

Page: 8

[y2]Add returned results information in the figure.